Open-Apple™

August 1986
Vol. 2, No. 8 7

ISSN 0885-4017
newstand price: $2.00
photocopy charge per page: $0.15

Releasing the power to everyone.

A 65802/65816 pre-boot

This month we're going to take an in-depth look at the newest members of the 6502 family, the 65802 and 65816 microprocessors. The 6502 was at the heart of the earliest Apple IIs, II-Pluses, and IIes. An enhanced version of this chip, known as the 65C02, is at the heart of IIcs and enhanced IIes. The 65802 is a direct replacement for these earlier chips—you can plug it into the 6502 socket of current versions of the Apple II. Meanwhile, Apple could do worse than to use the 65816 in future additions to the Apple II line.

Since I didn't know anything about these newer chips until just a few weeks ago, I commissioned Michael Fischer, author of *65816/65802 Assembly Language Programming*, published by Osborne/McGraw-Hill, to write an article for ***Open-Apple*** that would introduce the new members of the 6502 family to all of us. His article appears in this issue. Parts of it were taken directly from his book and appear with the permission of the publisher.

Before I turn things over to Fischer, however, let me supply your human operating systems with a little pre-boot information to ease your introduction to the new chips. Some of the stuff I'll cover Fischer also covers in his article, but it will probably help your understanding to hear the same story twice from different people.

Those of you who see nothing but obscure computer jargon in this month's issue should begin by going back and reading *The Magic of Peek and Poke* in the February issue (pages 2.2-2.5). That article is an introduction to microprocessors, memory, and how computers work. What it doesn't tell you is that inside old Queen 6502 there are some very special memory locations called *registers*. In the 6502 most of these registers can hold any number between zero and 255—they are one-byte, or 8-bit, registers. The most special one of all is called the A Register. A stands for *action*—and this is where it all takes place.

Other registers in the 6502 are the X and Y Index Registers, which are used for counting passes through loops and for accessing series of bytes in sequence; the Program Counter, which points to the byte that holds the machine language instruction currently being executed; the Status Register, which holds some 1-bit flags and switches; and the Stack Register, which points a crooked finger at our current position in the stack.

The stack, at least in the 6502, is a 256-byte data storage area that is kind of like a Rolodex flattened out and hung from the ceiling. The first number you push onto the stack goes on the card at the top. The next number you push goes onto the next card down, and so on. When you pull a number off the stack, you always get the last one pushed on. In other words, the last one on is the first one off. The Stack Register simply keeps track of the push/pull position inside this 256-byte stack.

Back in February, we learned that the memory inside an Apple II is most easily thought of as 256 *pages*, each holding 256 bytes—this makes 256 times 256 or 65,536 total bytes. The 6502 gives the very first of these pages, known as page zero, some special magical qualities.

Conquering intimidation. If microprocessor chips; their registers, stacks, and zero pages; their operation codes (instructions) and operands (the data to be used by the instruction) still intimidate you, it's because the people who write the original documentation for such chips don't concentrate hard enough on the words they use to describe things.

Frankly, my own first brushes with the 65816 left me cold and scared. The chip seemed vastly more complicated than the 6502. But, as Fischer will show you later, it's really just a 6502 at the core, surrounded by elegant flesh that gives it muscle. There are more registers in the 65816, yes, but the A register, and only the A register, is still where all the action is. The stack is still there, but it's bigger and there are lots more ways to use it. Likewise, page zero has become bank zero and its magic is more powerful.

After reading Fischer's article I finally figured out that my fear of this chip resulted from nothing but the words its designers used to describe it. The words are fear provoking—the chip itself is not.

The chip has what its designers call an *emulation mode*, in which it does what a 6502 can do just exactly like a 6502 does (although, even in this mode, it can also do lots more). It also has what's called a *native mode*, in which it does the things only the 65816 can do.

My problem is that the word "emulation" means a lesser thing acting like a greater thing. The 65816's designers, however, use the word to mean a greater thing (a 65816) acting like a lesser thing (a 6502). Thus, when I see the word, it takes my brain awhile to figure out what they really mean.

This wouldn't be so bad, except the word "native", which the 65816's designers use to describe the chip's enhanced mode, makes me think of a primitive state, not an advanced state. There are any number of word pairs (original-enhanced, front-back, outside-inside, startup-operational) that would have made better names for these modes than emulation-native.

Not even the chip's designers can change these words now, however, without creating a Babel. These are the words the designers handed down to us and these are the words we'll have to use to communicate with. They're the ones Fischer uses and other authors after him will use them, too.

But if the designers had worked as hard on the words they used to describe the chip as they worked on the rest of the chip's design, it would be an easier device to learn how to use. Countless hours of confusion and bug-hunting yet to come would have been avoided.

Ready for more examples? What does "reset" mean? If "set" means to make a bit equal to 1, shouldn't "reset" mean make equal to 1 a second time? But here "set" means make equal to 1 and "reset" means make equal to 0. This is an abomination. In the 6502 world we already had a perfectly

YOU KNOW THE GUY WHO BOUGHT ALL THAT SOFTWARE? HIS CHECK HAS A WARRANTY THAT SAYS IT'S TENDERED AS IS AND HAS NO FITNESS FOR ANY PARTICULAR PURPOSE, INCLUDING, BUT NOT LIMITED TO, CASHING.

good word, "clear," that meant make equal to zero. If everyone would use the word pair "clear-set," rather than "reset-set," the world would go round a good deal faster than it does.

Bringing this all up now is like arguing with the umpire. I don't do it because I think we can ever get better words for the 65816, but because I hope those of you who design chips (and other things) in the future will give lots of thought to the words you use to describe them.

The names of the 65816's two operational modes and our rebrush with set, incidentally, are just a little scare compared to the hand-wetting fear you'll feel when I tell you the names of some of the new chip's addressing modes. Fortunately there's a good solution to this problem, so don't let the following list frighten you away. Before I give you the solution, however, imagine eavesdropping on a discussion between two programmers who are deciding which of the following addressing modes to use:

```
direct indirect
direct indexed indirect
direct indirect indexed
direct indirect long
direct indirect long indexed
```

Not even certified HeDaPs could keep these straight. But by far the most confusing thing about the 65816's addressing modes is their names. Hundreds, maybe thousands, of people will give up on 65816 assembly language because they can't make any sense out of them. But it's no wonder — these names are nearly flat out nonsense to begin with. They are ten times worse than no names at all. Don't even try to remember or understand them. Instead, just associate what a command looks like — LDA (d),Y for example — with where the data is going to come from or go to.

For those of you who are new around assembly language, I should explain that an "addressing mode" is like a recipe used by a machine language instruction to find the data it's supposed to work on. For example, the 6502 has an *immediate mode*, which means the data is right there inside the program. LDA #3 tells the 6502 to load a 3 into the A Register. Then there's *absolute mode*, which means the data is at a certain address. LDA 3 tells the 6502 to load the A Register with the value in byte 3. And there's *indirect mode*, which means the data is at a final address that's being pointed to by an intermediate address. For example, LDA (3) tells the 6502 to look in bytes 3 and 4 for an address that points to another address, then to load whatever is at that other address into the A register. Finally, there's *indexed mode*, which means the data is at the given address plus the value in the X or Y Register. LDA 3,X, for example, gets the data at byte 8 if the X Register holds a 5. *All other addressing modes are nothing but combinations of these four basic types.*

The 6502 family has a large variety of addressing modes compared to most other chips. These modes are what give the chip much of its power. The toughest part of learning assembly language is the week of nightmares you have to toss and turn through to reconcile these silicon-based addressing modes with your organic brain. Once you're past that, however, assembly language programming is a snap.

256^3 = 65816. The first change you notice when moving from the 6502 to the 65816 is that where before you had 256 pages of memory to work with, you now have 256 *banks* of memory, each with 256 pages. The 65816 can use three full bytes — 24 bits — to specify a memory address. This gives it the ability to directly address 256 times 256 times 256 bytes of memory — 16,777,216 bytes (16 megabytes) in all.

The magical powers of the 6502's page zero are extended to all of bank zero on the 65816. Likewise, the 6502's 256-byte stack becomes a 65,536-byte stack.

The A, X, and Y registers on the 65816 can be configured as either 6502-like 8-bit registers (in which case a new, almost useless, 8-bit register called B appears) or as 16-bit registers. The A register and the index registers can be configured independently, so that, for example, X and Y can be 8 bits wide while A is 16 bits wide. When a register configured for 16-bits accesses memory, it gets two memory bytes — a full 16-bits — with one whack. For example, PHA pushes two bytes on the stack if A is configured for 16 bits. LDX $33 loads both byte $33 and byte $34 into X if X is configured for 16 bits.

The stack register is 16 bits wide on the 65816 — this is how the chip manages a 65,536 byte stack. The stack always lives in bank zero.

In order to directly access the full 16 megabytes of memory, the 65816 has two new registers. One is called the Data Bank Register and the other is called the Program Bank Register. The Program Bank Register always holds the bank number of the program that is currently executing. You can only change the Program Bank Register with certain new forms of the JMP and JSR (and RTS and RTI) instructions. All branches, such as BEQ and BCC, stay within the current bank, even if it means wrapping from byte $FFFF to byte $0000 in that bank. Thus the modules of a program can reside in several different banks, but any single module must be wholly within a single bank.

The Data Bank Register points to the bank that will be used by addressing modes that specify a two-byte address. There are also addressing modes that specify three-byte addresses — these can be used to get into any bank no matter what the contents of the Data Bank Register are.

There are also some addressing modes that use a one-byte address. In the 6502 these were called *zero page* addressing modes. The one-byte address was always assumed to be on page zero. On the 65816 these one-byte modes still exist, but they are called *direct* modes rather than zero page because they use a new register called the *Direct Register*. This register is 16-bits wide and is capable of pointing at any byte within bank zero. The one-byte address that follows the opcode is added to the Direct Register to calculate a two-byte address within bank zero. If you are using 8-bit X and Y registers, you can effectively have multiple 256-byte zero-pages anywhere within bank zero by manipulating the Direct Register. With 16-bit X and Y registers, on the other hand, you can access all of bank zero even if the Direct Register is left full of goose eggs.

There are four one-byte addressing modes on the 65816. The only memory bank that can be accessed with these modes is bank zero. Let's let *OPC* stand for an operation code that is valid with a given addressing mode. A capital *D* is the 16-bit Direct Register. A small *d* stands for a one-byte address following the opcode. The four one-byte modes are:

```
OPC d     D + d             "direct" (was "zero-page mode" on 6502)
OPC d,X   D + d + X         "direct indexed with X"
OPC d,Y   D + d + Y         "direct indexed with Y"
OPC d,S   S + d             "stack relative"
```

The *stack relative* mode is new and has all kinds of power. It makes it quite easy to pass variables to a subroutine using the stack, because the subroutine can dig the variables out of the stack without resorting to pushing, pulling, or otherwise modifying the stack. For example, here's a way for a calling routine to pass a value to a subroutine, and for the subroutine to pass a result back to the caller, without using any registers or absolute memory locations:

```
      PHA          Push value on stack.
      JSR CALC
      PLA          Pull result off stack.
      etc.

CALC  LDA 3,S      LDA 1,S would get the last value pushed
      etc.           on the stack. The JSR put two bytes there,
      STA 3,S        thus we use LDA 3,S.
      RTS          STA 3,S sends a result back to the caller.
```

Multiple values could be placed on the stack as long as the subroutine knew where to look for them with instructions such as LDA 5,S; LDA 7,S; and so on.

There are seven two-byte addressing modes. These modes can access any of the 256 possible memory banks. However, they use the contents of the Data Bank Register, also known as *DBR*, to specify the bank. Several of these modes are *indirect*. In the indirect examples that follow, the address in parantheses **points to** the two-byte address that is used for that mode. Note that in these cases only one address byte actually follows the opcode. This single byte is used to calculate the pointer. The pointer aims at the two-byte address the instruction actually uses. Where *a* stands for a two-byte address following an opcode, the seven two-byte addressing modes are:

```
OPC a       DBR + a           "absolute"
OPC a,X     DBR + a + X       "absolute indexed with X"
OPC a,Y     DBR + a + Y       "absolute indexed with Y"

OPC (d)     DBR + (D + d)     "direct indirect"
OPC (d),Y   DBR + (D + d) + Y "direct indirect indexed"
OPC (d,X)   DBR + (D + d + X) "direct indexed indirect"
OPC (d,S),Y DBR + (S + d) + Y "stack relative indirect indexed"
```

The (d,S),Y mode allows a subroutine to use a pointer that has been passed to it within the stack. Another possibility is to push pointers on the stack rather than using predetermined zero-bank locations. For example:

```
      LDY #0          clear Y
      LDA table.adr   get address of table
      PHA             push on stack
```

```
LDA (1,S),Y     get first value in table
PLY             pull address back off stack and discard
```

There are four three-byte address modes. Again, some of these modes actually have only one byte following the opcode. This single byte is used to calculate a pointer. The pointer is aimed at a three-byte address. These modes are called *indirect long* and use brackets rather than parantheses. Where *al* stands for a three-byte address following an opcode, the four modes are:

```
OPC al      al              "absolute long"
OPC al,X    al + X          "absolute long indexed with X"
OPC [d]     (D + d)         "direct indirect long"
OPC [d],Y   (D + d) + Y     "direct indirect long indexed"
```

There are several other addressing modes, but I better leave something for Fischer to talk about. On the next page there's a massive chart I have constructed that shows all the 65816 opcodes and addressing modes, and the addressing modes that can be used with each opcode. The numbers (1,2,3,4) indicate the total length of an instruction. All 65816 opcodes are one byte long. The other bytes in an instruction make up the operand. A single tick mark next to a number means that that opcode/addressing mode combination first appeared on the 65C02; a double tick mark indicates the combination is new on the 65816.

Study the chart for awhile, then read Fischer's article, then study the chart some more. The impressive thing is how everything we've learned using the 6502 can be leveraged into knowledge about the 65816. Knowing how to program the 6502 is the real pre-boot to the 65816. I'm sure we'll be talking about this chip much more in the future. I hope you find it as fascinating as I do.

Introduction to the 65802/65816

The 65816 ("sixty-five-eight-sixteen") is a microprocessor that is an upwardly compatible member of the 65xxx family of microprocessors. It can run programs that were written to run on a 6502/65C02. It is not a downwardly compatible chip, however—programs written to take advantage of the full power of the 65816, by and large, will not run on a 6502 or 65C02.

The 65816 was developed by the Western Design Center (2166 East Brown Road, Mesa AZ 85203). Their version of the chip is called a W65SC816. It is also manufactured by General Telephone and Electronics under a license from the Western Design Center. The GTE version is called a G65SC816.

The chip is an internal 16-bit microprocessor. However, its external data line is only 8 bits wide. It transmits 16 bits of data by first sending the low 8 bits of data (sometimes called the least significant byte) over the data line followed by the high 8 bits (sometimes called the most significant byte). This technique of sending different kinds of information over the same line at different times is called multiplexing. While the 65816 transmits data in two 8-bit chunks, a programmer needs only to give it one command to transmit a 16-bit chunk of data.

Two new chips. There are actually two flavors of this new 16-bit processor. The 65802 ("sixty-five-eight-oh-two") is similar to the 65816 in that it handles 16-bit data. It has, however, a 16-bit address line, which means it can directly address only 65,536 bytes—the same as the 6502/65C02. By contrast, the 65816 can work with either 16-bit or 24-bit addresses, permitting it to address up to 16 megabytes.

When using its 24-bit addressing capability, the 65816 multiplexes the upper 8 bits of the address on the data bus while it puts the lower 16 bits on the address bus. When using 16-bit addressing, the 65816 uses the address bus alone.

At first glance it would seem that there is little reason for the existence of the 65802, since the 65816 can do everything it can do and address up to 16 megabytes of memory besides. The advantage of the 65802 is that it is "pin compatible" with the 6502/65C02. This means that each pin on the 65802 chip performs the same function as the similarly placed pin on the 6502/65C02—in other words, a 65802 can be placed in the same socket on an Apple IIe as the 6502/65C02 and give you the advantage of the extended assembly language instruction set. This will only be an advantage to those who do assembly language programming. A 65816 will not work if placed in the 6502/65C02 slot on an Apple IIe.

From now on the term "65816," as used in this article, also includes the 65802 except in discussions of 24-bit addressing and other areas where the context makes it clear that only the 65816 is being discussed.

Emulation mode. How does the 65816 accomplish its magic of working as a 6502, 65C02 and a 65816? The chip can run in either native mode or emulation mode. In emulation mode it responds to 6502 commands nearly the same as a 6502. The important operational differences from a standard 6502/65C02 are:

1. All 65816 operation codes and addressing modes function, although the upper 8 bits of 24-bit addresses are ignored. Programs that use opcodes that were undefined on the 6502/65C02 will be sorry on the 65816.

2. There is a second accumulator, known as the B register. The only way to use this register is to exchange its contents with the A register by means of the XBA (eXchange B and A registers) command. The command is also known as the SWA (SWAp) command.

3. The Stack Pointer in the 6502/65C02 is an 8-bit register that permanently points to an address on page one of memory (from $0100 - $01FF). The value in the pointer is between $00 and $FF. The 65816's Stack Pointer is a 16-bit register that can point to any address between $0000 and $FFFF. While the 65816 Stack Pointer in emulation mode normally works with stack values between $0100 and $01FF, several different operations can increment or decrement the Stack Pointer beyond this range when transferring more than one byte of data. All of the operations that can do this are new to the 65816 and would not normally be used in emulation mode, however.

4. The Direct Register is a 16-bit register that is new to the 65xxx family with the 65816. As long as its value is $0000 it has no effect on emulation mode operation with 6502/65C02 commands. However, if it is set to values other than zero it can have a profound effect on emulation mode because it effectively moves page zero elsewhere. It is extremely important that programs that modify the Direct Register clear it back to zero before quitting.

5. The TSC command, new with the 65816, transfers the contents of the Stack Pointer to the C register (a new name for the A register when it's in 16-bit mode). However, in emulation mode (as in native mode), TSC transfers not only the lower 8 bits of the Stack Pointer to the accumulator, it also transfers the upper 8 bits (permanently set to $01 in emulation mode) to the B register. This effect is only significant if you are storing a value in the B register.

Native mode. That's enough emulation. Now let's look at the architecture of the 65816 in native mode. First a general description. The 65816 contains the following registers:

1. An accumulator that can be configured as either a 16-bit accumulator (known as C or A) or as two 8-bit accumulators (known as A and B). Configuration takes place by placing a 1 (for 8-bit mode) or a 0 (for 16-bit mode) in a status bit (known as the M bit) in the 65816's processor Status Register. (More on that later.)

2. Two index registers, called X and Y. The registers can be configured as either 16- or 8-bit registers by placing a 1 (for 8-bit mode) or a 0 (for 16-bit mode) in the new X bit of the Status Register.

3. A 16-bit Stack Pointer that permits the setting of the stack anywhere in the first bank of memory (locations $000000-$00FFFF).

4. A 16-bit Program Counter that holds the address of the next command.

5. A 16-bit Direct Register that permits the addressing of memory anywhere in the first bank of memory faster than addressing memory located elsewhere.

6. A Status Register consisting of 9 bits, 8 of which are directly accessible. This register is discussed in greater detail below.

7. There are two 8-bit registers in the 65816 that give it the capacity for full 24-bit addressing. These two registers are also present in the 65802 but they have no effect on the actual address since that processor is restricted to 16-bit addressing. The Data Bank Register, which can hold the upper 8 bits of a 24-bit data address, is used only in certain long addressing modes discussed later in this article. The Program Bank Register is used to hold the upper 8 bits of the address of the next program command. Setting the Program Bank Register is restricted to a very few commands, as discussed below.

The Status Register. As discussed earlier, the Status Register is a collection of various bits. These bits consist of some mode-select bits (which, for instance, determine whether mathematical operations take place in binary or decimal mode), and some status flags that indicate what occurred as the result of some recent operation. Some of these bits are unchanged from the 6502/65C02. These include:

1. The carry (C) bit, a status flag, which indicates whether the latest arithmetic or logical command resulted in a carry out of the eighth or sixteenth bit.

65802/65816 OPERATIONS AND ADDRESSING MODES

The numbers 1,2,3,4 indicate the total length of the instruction and its operand in bytes. Combinations marked ′ first appeared on the 65C02, ″ on the 65816.

	bank access notes-->			bank 0 only DBR not used				all banks DBR + 2-byte adr			all banks 3-byte adr		all banks DBR + 2-byte adr				all banks 3-byte adr		see blw	current bank only can't change PBR			
addressing modes	A	i	s	#	d	d,X	d,Y	d,S	a	a,X	a,Y	al	al,X	(d)	(d),Y	(d,X)	(d,S),Y	[d]	[d],Y	xyc	r	rl	(a)	(a,X)
ARITHMETIC, LOGICAL, COMPARISON, AND MEMORY ACCESS COMMANDS																							
LDA/ADC/SBC/CMP				2/3″	2	2		2″	3	3	3	4″	4″	2′	2	2	2″	2″	2″				
AND/ORA/EOR				2/3″	2	2		2″	3	3	3	4″	4″	2′	2	2	2″	2″	2″				
STA					2	2		2″	3	3	3	4″	4″	2′	2	2	2″	2″	2″				
BIT				2′/3″	2	2′			3	3′													
ASL/LSR	1				2	2			3	3													
ROL/ROR	1				2	2			3	3													
INC/DEC	1′				2	2			3	3													
TSB/TRB					2′				3′														
LDX				2/3″	2		2		3		3	Immediate mode commands require 2- or 3-byte operands depending on											
LDY				2/3″	2	2			3	3		whether the register being used is set for 8 or 16 bits.											
STX					2				3														
STY					2	2			3														
CPX/CPY				2/3″	2				3														
INX/DEX/INY/DEY	1																						
STZ					2′	2′			3′	3′	Stores a zero in memory without changing the A Register.												
MVP/MVN					Move block: A=size of block-1, X=source, Y=dest, xyc=source bank, dest bank. 3″																		
					Use MVP if dest>source and blocks overlap. Use MVN if source>dest. If no overlap, either command is ok.																		
COMMANDS THAT TRANSFER DATA FROM REGISTER-TO-REGISTER																							
TAX/TAY/TXA/TYA	1																						
TXS/TSX	1																						
TXY/TYX	1″																						
TCS/TSC	1″				C indicates the A Register in 16-bit mode.																		
TCD/TDC	1″				D indicates the new Direct Register.																		
XBA	1″				B is the high 8 bits of the 16-bit A Register. A indicates the low 8 bits. XBA moves the contents of A to B and B to A.																		
COMMANDS THAT CHANGE THE STATUS REGISTER (also known as P)																							
CLC/SEC	1																			N is sign 1=neg Z is zero 1=true			
CLD/SED	1				6502 mode N V 1 B D I Z C 1 is unused; B is break,1=true															V is overflow 1=true C is carry 1=true			
CLI/SEI	1				65802/65816 mode N V M X D I Z C M is A-reg width, 1=8 bits, 0=16 bits															D is BCD bit 1=true			
CLV	1				X is X & Y width, 1=8 bits, 0=16 bits															I is IRQ bit 1=interrupts disabled			
REP/SEP				2″	REP clears, SEP sets the bits indicated--e.g. REP %00000001 is the same as CLC.																		
XCE	1″				XCE exchanges Carry Bit with Emulation Bit. E=1 for 6502 mode, E=0 for 802/816 mode.																		
COMMANDS THAT WRITE TO AND READ FROM THE STACK																							
PHA/PLA				1																			
PHP/PLP				1																			
PHX/PLX/PHY/PLY				1′																			
PHD/PLD				1″	D is the new Direct Register																		
PHB/PLB				1″	B is the new Data Bank Register																		
PHK				1″	K is the new Program Bank Register--note that this register can only be read, not written to.																		
PEA				3″	PEA is Push Immediate--the 2-byte operand is pushed on the stack.																		
PEI				2″	PEI is Push Indirect--the 1-byte operand is added to the Direct Reg; the 2-byte value at that address is pushed on the stack.																		
PER				3″	PER is Push Relative--the 2-byte operand is added to the Program Counter and the result is pushed on the stack.																		
PROGRAM CONTROL COMMANDS																							
BPL/BMI																				2			
BVC/BVS					All branches except BRL can go forward 129 bytes or backward 126 bytes.															2			
BNE/BEQ					BRL can go forward 32770 bytes or backwards 32765 bytes. However, all															2			
BCC/BCS					branches stay within a single bank. They always wrap from $FFFF to															2			
BRA					$0000 within the current bank.															2′			
BRL																					3″		
JSR									3														3″
JMP									3													3	3′
RTS/RTI				1																			
JSL												4″											
JML												4″										3″	
RTL				1″	Only 5 commands can change the contents of the Program Bank Register--JSL al, JML al, JML (a), RTL and RTI.																		
BRK				1/2″																			
OTHER COMMANDS																							
NOP				1																			
STP				1′	Stop until reset.																		
WAI				1′	Wait until interrupt.																		
COP				2″	For use with a co-processor.																		
WDM				1″	Reserved for activating future 32-bit chip.																		

2. The zero (Z) bit, a status flag, which indicates if the result of the last operation of any kind was zero.

3. The interrupt disable (I) bit, a mode-select bit, which can enable or disable interrupt requests.

4. The decimal (D) bit, a mode-select bit, which determines if the mathematical operations will occur in binary or binary-coded-decimal.

5. The overflow (V) bit, a status flag, which indicates whether an overflow resulted from the most recent mathematical operation. An overflow is defined as a carry from bit 14 to 15 (or from bit 6 to 7 in 8-bit mode).

6. The negative (N) bit, a status flag, which indicates whether there was a negative result from the most recent mathematical or logical operation. A negative result is defined as a value with the most significant bit (bit 15 in 16 bit mode) set to one.

One bit familiar to 6502/65C02 programmers that is not present in the 65816 native mode Status Register is the break (B) bit. It is present, however, when the 65816 is in emulation mode. This status flag indicates whether the cause of an interrupt was hardware or software. This bit is not really needed with the 65816, as we'll see later, in the discussion on interrupts.

Three bits of the Status Register are new with the 65816. These are:

1. The index register (X) bit, a mode-select bit, which determines whether the data width of the X and Y index registers is 8 (X=1) or 16 (X=0) bits.

2. The memory select (M) bit, a mode-select bit, which determines whether the data width of the accumulator and commands such as STZ is 8 (M=1) or 16 (M=0) bits.

3. The emulation (E) bit, a mode-select bit, which determines whether the 65816 is in emulation (E=1) or native (E=0) mode.

All of the 6502/65C02 mode-select bits and some of the 6502/65C02 status flags can be set and cleared by individual commands in either native or emulation mode. For example, the carry bit can be set by the SEC (for SEt Carry bit) command and the interrupt disable bit can be cleared by the CLI (for CLear Interrupt disable) command. But there are some bits for which there are no set or clear commands.

None of the new 65816 status bits have their own set or clear commands. Instead there are two new commands, REP (for REset Processor status bits) and SEP (for SEt Processor status bits), which can clear or set any combination of the Status Register's bits except the emulation bit.

The emulation bit is not actually one of the 8 bits that make up the Status Register. Instead, it conceptually hides behind the carry bit and can be set or cleared by a single command, XCE (for eXchange Carry and Emulation bits). The XCE command swaps the values in the carry and emulation bits. Thus to set the emulation bit (and place the 65816 into emulation mode) you would give the two commands SEC (SEt Carry bit) and XCE, resulting in a set emulation bit. To clear the emulation bit (and place the 65816 into native mode) you would give the two commands CLC (CLear Carry bit) and XCE, resulting in a cleared emulation bit.

Interrupts. An interrupt is a signal to the microprocessor that tells it to stop what it's doing and attend to something else. The 6502/65C02 supports four types of interrupts; the 65816 supports six. All six are supported in both emulation and native modes. For each type of interrupt there is a fixed location in memory called an *interrupt vector*, which points to a routine for handling that type of interrupt.

Of the 6502's four interrupts, three (IRQ, interrupt request; NMI, non-maskable interrupt; and Reset) are generated by electronic signals from other hardware and the fourth is caused by the BRK (break) command. BRK and IRQ share an interrupt vector—the 6502 (and the 65816 in emulation mode) differentiates between them with the Status Register's B bit.

In 65816 native mode, on the other hand, BRK and IRQ are separated so that a BRK bit is no longer necessary. One of the two new interrupts is Abort, which can be used by special types of hardware to fool the 65816 into thinking there is more RAM in the computer than is really there. The other is a software interrupt called COP (co-processor), which allows a program to call special-purpose auxiliary processors. Here is a complete list of the interrupt vectors the 65816 supports:

65816 interrupt vectors (all vectors are in bank 0, page $FF)

	emulation mode vectors	native mode vectors	
IRQ	FE-F	EE-F	
Reset	FC-D	reserved (uses emulation vector)	
NMI	FA-B	EA-B	
Abort	F8-9	E8-9	for implementing virtual memory
BRK	reserved (uses IRQ)	E6-7	
COP	F4-5	E4-5	for implementing co-processing

The effect of a reset in the 65816 is to set the Direct, Data Bank, Program Bank, X high and Y high Registers to zero; set the Stack Pointer high to $01; clear the D bit and set the M, X, I and E bits in the Status Register. This puts the 65816 in emulation mode—for this reason there is no separate native mode reset vector.

Let's take a BRK. The BRK instruction is considered a single byte instruction in the 6502/65C02. The 65816, by contrast, treats both the BRK and the COP instructions as two-byte commands. The byte following the command itself serves different purposes for the BRK and COP instructions:

1. When BRK is used in debugging, a common use, it is often inserted in place of a 2-byte instruction. The BRK code itself ($00) is placed over the first byte and the second byte is skipped. A return from interrupt (RTI) instruction will pick up at the proper location following a two-byte BRK instruction.

2. The 1-byte operand following the COP code is known as a signature byte. This byte can be used to pass parameters to the interrupt-handling routine and thus to the co-processor. Signature byte values $80-$FF are reserved and $00-$7F are available for programmer use.

There are two special-purpose software instructions that also work with interrupts. The STP (SToP the clock) and WAI (WAit for Interrupt) commands place the 65816 into a state of hibernation until a hardware interrupt resumes normal operation. The STP command's wait state can be terminated only by a reset. The WAI command's wait state can be terminated by a reset, NMI, or IRQ.

Addressing modes. An assembly language program on most microprocessors, including all members of the 65xxx family, consists of commands and, in most cases, data upon which the commands operate. The data sometimes follows the command in the program; other times the information following the command shows where in the computer's memory the data is located. The various methods by which the microprocessor finds the information are referred to as addressing modes.

The 6502 has 14 addressing modes. The 65C02 has all 14 of these modes plus two additional ones. All 16 of these modes are also found on the 65816, although five of them—the zero-page addressing modes—are modified. On the 6502/65C02, the zero-page addressing modes use a single-byte value that is the location of the data on page zero ($0000-$00FF). On the 65816, the zero-page addressing modes are called direct modes. Direct addressing consists of a single-byte value that follows the instruction. The value is added to the 16-bit value in the Direct Register and the result is an address in bank zero ($000000-$00FFFF).

In emulation mode, if the low byte of the Direct Register is zero, direct page operations take place on the page pointed to by the high byte of the Direct Register. This is known as a relocatable zero page. Operations that extend beyond the boundary of the page will wrap around inside the page. For example, LDA $F0,X, with X=$20 and D=$0800, will access byte $0810.

If the low byte of the Direct Register is a value other than zero (in other words, if the relocated zero page doesn't lie on a page boundary), on the other hand, operations that extend beyond the boundary of the page will NOT wrap. For example, LDA $F0,X, with X=$20 and D=$0880 will access byte $0990. The main cause of a non-zero direct register in emulation mode is a failure to clear it to zero before leaving native mode.

The 65816 adds eight new forms of addressing to the ones that already existed under the 65C02. Many of the added modes are simply long forms of existing 6502 modes. A long form of addressing is 3 bytes long, utilizing the 24-bit addressing capabilities of the 65816. The long forms of addressing include:

1. Absolute long

2. Direct indirect long indexed

3. Absolute long indexed with X (there is no Absolute long indexed with Y)

4. Direct indirect long

An additional special form of long addressing is the program counter relative long mode. On the 6502/65C02 there are several branch instructions, using a one-byte displacement, which can modify the location of program execution up to 129 bytes ahead of the present location or 126 bytes behind.

With the program counter relative long mode, there is a two-byte displacement. This results in a change in the execution address of as many as 32,770 bytes ahead or 32,765 bytes behind. Only one branch instruction (BRL) supports this addressing mode, however, and it branches always.

Two additional new addressing modes use the expanded 16-bit stack pointer for part of the memory address used. The stack relative addressing mode takes the one-byte value following the command and adds it to the value in the stack pointer to obtain the actual memory location of the data, which is always in bank zero of memory. The stack relative indirect indexed addressing mode takes the one-byte value following the command and adds it to the value of the Stack Pointer to obtain an address in bank zero. That address contains a two-byte value that is the lower two bytes of the actual data address. The Data Bank Register contains the value of the highest byte of the 24-bit address. The Y Register is used as an index from that address.

The final new addressing mode is used with the new block move command and is discussed with that command.

Commands. The various instructions that can be given to the 65816 are known as opcodes or mnemonics. Each instruction consists of a three-letter abbreviation that somewhat explains the function of the instruction.

There is no universally-agreed upon method of categorizing instructions. The organization I will use here is loosely based on the Microprocessor Assembly Language Draft Standard of the IEEE. Differences in the command sets of the 65xxx family of microprocessors will be discussed within this organization.

The 6502 has 56 commands and 14 addressing modes that can be combined into 151 different legitimate instructions. The 65C02 adds 10 new commands, two new addressing modes, and 29 additional legitimate combinations. The 65816 adds 26 new commands (for a total of 92), eight new addressing modes (a total of 24), and 76 new legitimate combinations (a total of 256).

Arithmetic logic instructions. These commands take a value found in a specified location and perform either an arithmetic (add, subtract, increment, or decrement), logical (and, or, or exclusive-or), or shift/rotate operation on it. The 65816 adds no new instructions that were not already found on the 65C02, although it adds about 30 new command/address mode combinations, resulting from either the new long or stack relative addressing modes.

ARITHMETIC LOGIC INSTRUCTIONS

Command	6502 Forms	65C02 Forms	65816 Forms	function
ADC	8	1	6	add memory to accumulator with carry
AND	8	1	6	AND memory with accumulator
ASL	5	0	0	shift one bit left, memory or accumulator
DEC	4	1	0	decrement memory or accumulator by one
DEX	1	0	0	decrement X register by one
DEY	1	0	0	decrement Y register by one
EOR	8	1	6	Exclusive OR memory with accumulator
INC	4	1	0	increment memory or accumulator by one
INX	1	0	0	increment X register by one
INY	1	0	0	increment Y register by one
LSR	5	0	0	shift one bit right, memory or accumulator
ORA	8	1	6	OR memory with accumulator
ROL	5	0	0	rotate one bit left, memory or accumulator
ROR	5	0	0	rotate one bit right, memory or accumulator
SBC	8	1	6	subtract memory from accumulator with borrow
TRB	0	2	0	test and reset bit
TSB	0	2	0	test and set bit
---	--	--	--	
TOTAL	72	11	30	

Branch/Jump instructions. Normally the 65816 executes machine code in sequential order. The branch and jump instructions send the 65816 to a new location in memory for the next instruction. The 65816 adds four of these program control instructions to those available on the 65C02—all of which result from long addressing modes. These are a long JMP using 24-bit addressing, a long JSR using 24-bit addressing, a long branch using 16-bit displacement, and a long return from subroutine using 24-bit addressing.

BRANCH/JUMP INSTRUCTIONS

Command	6502 Forms	65C02 Forms	65816 Forms	
BCC	1	0	0	branch if carry clear
BCS	1	0	0	branch if carry set
BEQ	1	0	0	branch if equal
BMI	1	0	0	branch if minus
BNE	1	0	0	branch if not equal
BPL	1	0	0	branch if plus
BRA	0	1	0	branch always
BRL	0	0	1	branch always, long
BVC	1	0	0	branch if overflow clear
BVS	1	0	0	branch if overflow set
JML	0	0	2	jump, long
JMP	2	1	0	jump
JSL	0	0	1	jump to subroutine, long
JSR	1	0	1	jump to subroutine
RTI	1	0	0	return from interrupt
RTL	0	0	1	return from subroutine, long
RTS	1	0	0	return from subroutine
---	--	--	--	
TOTAL	13	2	6	

Data transfer instructions. Perhaps the most commonly used assembly language instructions are the data transfer commands, which move a value from one location inside the computer to another. These locations may be in the computer's RAM, its ROM, or a 65816 register. The 65816 adds nine new data transfer instructions. These additional instructions are among the most significant of the new 65816 commands. They include:

1. The block move commands (MVN and MVP), which move a block of up to 65,536 bytes from any memory location to any other memory location. Before giving the command, the X Register must contain the lower 16 bits of the source address, the Y Register must contain the lower 16 bits of the destination address, and the Accumulator must contain one less than the number of bytes to be moved. The format of the instruction consists of the opcode followed by the source bank number (8 bits) and the destination bank number (8 bits).

2. New inter-register transfer commands, which move data between the Accumulator and either the Stack Pointer or the Direct Register, and between the X and Y Registers. Some of these instructions, and the previous inter-register transfer commands, function differently depending on the setting of the M and X bits. In general, if the contents of an 8-bit register are transferred to a 16-bit register, a zero will be placed in the high byte of the 16-bit register. If a 16-bit register is transferred to an 8-bit register, the high byte will be lopped off. However, transfers between the Accumulator and Stack Pointer and between the Accumulator and Direct Register are, with one exception, always 16-bit transfers (the B register is used for the high byte if A is configured for 8 bits). The only exception is a transfer of the Accumulator to the Stack Register in emulation mode, in which case the contents of B will be disregarded and the high byte of the Stack Pointer will remain $01.

3. The intra-register transfer command, XBA, which exchanges the value in the A and B portions of the 16-bit accumulator. Note that the other transfer commands transfer data from one register to another. At the end of the transfer both registers hold the *same* value. This command swaps the register values. At the end of the exchange the registers hold *opposite* values.

DATA TRANSFER INSTRUCTIONS

Command	6502 Forms	65C02 Forms	65816 Forms	
LDA	8	1	6	load accumulator with memory
LDX	5	0	0	load X register with memory
LDY	5	0	0	load Y register with memory
MVN	0	0	1	move block, negative
MVP	0	0	1	move block, positive
STA	7	1	6	store accumulator in memory
STX	3	0	0	store X register in memory
STY	3	0	0	store Y register in memory
STZ	0	4	0	store zero in memory
TAX	1	0	0	transfer accumulator to X register
TAY	1	0	0	transfer accumulator to Y register
TCD	0	0	1	transfer C accumulator to direct register
TCS	0	0	1	transfer C accumulator to stack register
TDC	0	0	1	transfer direct register to C accumulator
TSC	0	0	1	transfer stack register to C accumulator
TSX	1	0	0	transfer stack register to X register
TXA	1	0	0	transfer X register to accumulator
TXS	1	0	0	transfer X register to stack register
TXY	0	0	1	transfer X register to Y register
TYA	1	0	0	transfer Y register to accumulator
TYX	0	0	1	transfer Y register to X register
XBA	0	0	1	exchange B and A accumulators
	--	--	--	
TOTAL	37	6	21	

Stack instructions. Some commands either place data on the stack (called pushing) or remove it from the stack (called pulling). Two new instructions pull data from the stack into the Data Bank Register (PLB) and the Direct Register (PLD). Three instructions push the value in registers onto the stack—PHB (Data Bank Register), PHD (Direct Register) and PHK (Program Bank Register). Note that there is no pull Program Bank Register command. The only commands that can change the value in the program bank register are JSL al, JML al, JML (a), RTL, and RTI.

Three additional stack instructions push data that are not found in 65816 registers. The PEA (Push Effective Address or immediate data) instruction places the 16-bit value that follows the command onto the stack. The PEI (Push Effective Indirect address) instruction adds the value of the direct register to the one-byte value following the instruction to obtain an address. It then places the 16-bit value found at that address on the stack.

Potentially the most significant of the new 65816 instructions is the PER (Push Effective program counter Relative data) command. This instruction adds the value of the two-byte operand that follows to the current value of the program counter and places the resulting value on the stack. This instruction provides the potential of true position-independent code that doesn't require the tricks needed with the 6502/65C02.

STACK INSTRUCTIONS

Command	6502 Forms	65C02 Forms	65816 Forms	
PEA	0	0	1	push effective absolute address on stack
PEI	0	0	1	push effective indirect address on stack
PER	0	0	1	push effective PC relative adr on stack
PHA	1	0	0	push accumulator on stack
PHB	0	0	1	push data bank register on stack
PHD	0	0	1	push direct register on stack
PHK	0	0	1	push program bank register on stack
PHP	1	0	0	push status register on stack
PHX	0	1	0	push X register on stack
PHY	0	1	0	push Y register on stack
PLA	1	0	0	pull accumulator from stack
PLB	0	0	1	pull data bank register from stack
PLD	0	0	1	pull direct register from stack
PLP	1	0	0	pull status register from stack
PLX	0	1	0	pull X register from stack
PLY	0	1	0	pull Y register from stack
---	--	--	--	
TOTAL	4	4	8	

Status instructions. Three new commands permit modification of the processor status byte. These are REP, SEP and XCE. They were discussed earlier in this article.

STATUS INSTRUCTIONS

Command	6502 Forms	65C02 Forms	65816 Forms	
CLC	1	0	0	clear carry flag
CLD	1	0	0	clear decimal mode
CLI	1	0	0	clear interrupt disable bit
CLV	1	0	0	clear overflow flag
REP	0	0	1	reset status bits
SEC	1	0	0	set carry flag
SED	1	0	0	set decimal mode
SEI	1	0	0	set interrupt disable status
SEP	0	0	1	set status bits
XCE	0	0	1	exchange carry and emulation bits
---	--	--	--	
TOTAL	7	0	3	

Test instructions. No new commands are added to this category by the 65816. However six new addressing mode/command combinations were added to the CMP command, resulting from the new long and stack relative modes.

TEST INSTRUCTIONS

Command	6502 Forms	65C02 Forms	65816 Forms	
BIT	2	3	0	bit test
CMP	8	1	6	compare memory and accumulator
CPX	3	0	0	compare memory and X register
CPY	3	0	0	compare memory and Y register
---	--	--	--	
TOTAL	16	4	6	

Miscellaneous instructions. Some commands defy categorization. The 65816 adds two new instructions. It also includes two relatively unknown commands that appear only on the Western Design Center version of the 65C02, WAI (wait for interrupt) and STP (stop until Reset). The new instructions are COP, which is discussed in this article under interrupts, and WDM, which is a reserved instruction that will be used in the future to provide 32-bit floating point math and data operations on the 65832 processor, which Western Design is now working on. This chip will support all features of, and will be pin-compatible with, the 65816.

MISCELLANEOUS INSTRUCTIONS

Command	6502 Forms	65C02 Forms	65816 Forms	
BRK	1	0	0	force break
COP	0	0	1	coprocessor
NOP	1	0	0	no operation
STP	0	1	0	stop the clock
WAI	0	1	0	wait for interrupt
WDM	0	0	1	reserved for 65832
---	--	--	--	
TOTAL	2	2	2	

Assemblers. There are three Apple II assemblers that can handle 65816 code:

1. *Merlin Pro* is a macro assembler that runs on the Apple IIe and IIc. Two versions come in one package; one version runs under DOS 3.3 and the other under ProDOS. Other goodies with the package include a disassembler to aid you in building source code files from object code, a linker to generate relocatable code, and a commented, disassembled source code listing of the Applesoft interpreter.

Merlin Pro is the easiest of the three assemblers to use and is the least esoteric in its pseudo opcodes. Its major disadvantage is that it presently does not support 24-bit addressing directly, although you can write a macro to accomplish the same thing. Glen Bredon, the author of *Merlin*, has added 24-bit addressing to a new version of the assembler presently in beta test.

The publisher of *Merlin Pro* is Roger Wagner Publishing, Inc., 10761 Woodside Avenue, Suite E, Santee, CA 92071 (619) 562-3670. It sells for $99.95.

2. *Orca/M* (try reading it backward) is the most powerful of the 65816 assemblers. It handles 24-bit addressing and includes a linker that creates relocatable code. It also has a disassembler and a macro library with routines for mathematics, input and output, graphics, and other miscellaneous functions. *Orca/M* adheres most closely to the assembler specifications of the Western Design Center, creators of the 65816.

The power of *Orca/M* also results in its complexity. Its use of pseudo opcodes, particularly for macros and conditional assembly, is the least like the assemblers common to the Apple world. Indeed, *Orca/M* comes with an operating system, a shell around ProDOS, that allows the adding of functions to the assembly environment. It is a struggle to get used to; whether it is worth the struggle depends on whether you need its power.

The publisher of *Orca/M* is The Byte Works, Inc., 8000 Wagon Mound Drive NW, Albuquerque, NM 07120 (505) 898-8183. It sells for $79.95.

3. The *S-C Macro Assembler* comes with an assembler that can handle 6502, 65C02, Sweet-16 and 65816 code with full 24-bit addressing. The editor is nearly as easy to use as the *Merlin Pro* editor. Its macro, conditional assembler, and other pseudo opcodes are easy to understand although a bit different than the Apple "standard." It comes in either a DOS 3.3 or a ProDOS version (or you can purchase both versions together at a substantial discount).

A significant advantage to the *S-C Macro Assembler* is that it is part of a system. Various add-on programs are available including a cross-reference utility, a full screen editor, a commented Applesoft disassembly, and even cross assemblers to create 6800, 6809, 68000, Z-80, and PDP-11 code. You can also purchase the source code for the assembler itself, a commitment to open programming rarely seen.

The biggest advantage of the system, though, is a monthly periodical distributed and largely written by the assembler's creator, Bob Sander-Cederlof. *Apple Assembly Lines* is the only publication today carrying any significant 65816 assembly language articles. The assembler used for program listing in Apple Assembly Lines is the *S-C Assembler.*

The publisher of the *S-C Macro Assembler* is S-C Software Corporation, 2331 Gus Thomasson, Suite 125, P.O. Box 280300, Dallas, TX 75228 (214) 324-2050. It sells for $100 for either the DOS 3.3 or ProDOS version and $120 for both versions.

Hardware. There are several different paths you can follow to put either a 65802 or a 65816 in your Apple.

1. The least expensive way is to remove the 6502/65C02 and replace it with a 65802 chip. This method restricts you to 16-bit addressing. The chip is available from S-C Software for $50.

2. Apple IIe owners can use an Apple16 65816 Co-Processor Board. This board fits into any available slot of the Apple IIe and has a 65816 chip with 256K of linearly addressable memory. Of course you will have to create you own routines to view the contents of memory above the first 64K as the Apple IIe Monitor only works on the first 64K bank. The board is available from the Com Log Corporation, 11056 N. 23rd Drive, #104, Phoenix, AZ 85029 (602) 248-0769. It costs $395.00.

3. Apple IIe owners can also purchase a card that fits onto the motherboard, not into an expansion slot. Checkmate Technology, Inc., 509 South Rockford Drive, Tempe, AZ 85281 (602) 966-5802, makes a MultiRAM EX 65816 Co-Processor Card. A similar card is also made by Applied Engineering, P.O. Box 798, Carrollton, TX 75006 (214) 241-6060. For both cards, installation consists of removing the 6502/65C02 and the MMU chips from your Apple IIe motherboard, placing the MMU chip onto the 65816 card, and placing the pins on the 65816 card into the now empty 6502/65C02 and MMU sockets. Each card can also be attached, via a supplied cable, to the respective expanded 80 column card (Multiram IIe by Checkmate Technology and Ramworks II by Applied Engineering) to permit linear addressing of 256K and more of memory. The Applied Engineering board can also connect to the Applied Engineering RamFactor standard slot memory expansion board. Prices: MultiRam EX, $189; MultiRam IIe, $159.95 (with 64K); Applied Engineering 65C816 16 Bit Card, $159; RamWorks II, $179 (with 64K); RamFactor, $239 (with 256K).

4. Checkmate Technology also makes a card similar to the MultiRAM EX that works with their CX board, an Apple IIc memory expansion system. This is the only method by which Apple IIc owners can put a 65816 in their computer. Prices: 65C816 upgrade for CX, $119.95; CX board, $199.95 (with 256K).

5. There is a "16-bit option" advertised for the Applied Engineering Transwarp fast processor board for the Apple IIe. This 16-bit option is a 65802, not a 65816. Prices: 16-bit 65802 upgrade, $89; Transwarp, $279.

Books. There are four books announced on assembly language programming for the 65816. They are listed here in alphabetical order. No attempt is made at reviewing these books both because not all are currently available and because I might be considered somewhat biased on this subject.

1. *65816/65802 Assembly Language Programming* by Michael Fischer, published by Osborne/McGraw-Hill. Currently available. $19.95.

2. *Programming the 65816* by David Eyes and Ron Lichty, to be published by Brady Communications. Not currently available. $22.95.

3. *Programming the 65816* by William Labiak, published by Sybex. Currently available. $22.95.

4. *The Handbook: 6502, 65C02 and 65816* by Stephen Hendrix, published by Weber Systems. Availability unknown. $17.95.

Ask (or tell) Uncle DOS

Whoa, looks like Fischer and I got so long-winded there's almost no space left this month for letters. Next month Uncle DOS will get at least six pages to make up for the shortage this month, however, so send him something good today.

Hard copy

I am very interested in the hardware side of the Apple and was wondering if there are any newsletters or magazines devoted to Apple hardware?

Robert T. Muir
Ferndale, Calif.

I don't know of any newsletters or magazines devoted completely to Apple hardware. I assume you have read the two books ***Understanding the Apple II*** *and* ***Understanding the Apple IIe****, by Jim Sather ($22.95 and $24.95 from Quality Software, 21610 Lassen St #7, Chatsworth, CA 91311). These are to Apple hardware what the old and new testaments are to Christian theology.*

AppleWorks DIFficulties

I recently attempted to move a lengthy *VisiFile* data base to AppleWorks, using the DIF conversion program provided by *VisiFile*, Apple's CONVERT program, and the AppleWorks "make a new file from a DIF file" option. This didn't work—AppleWorks aborted the read function immediately after displaying the "Getting this file" message.

I developed a really ugly work-around by treating the DIF file as a text file and editing out all sorts of garbage (you wouldn't want to know the details), but would appreciate any tips on why the prescribed procedure did not work and how it might be fixed.

Douglas J. Sietsema
Culver City, Calif.

I don't know anything about ***VisiFile****, but I think I know enough about DIF files and about AppleWorks to answer your question. DIF files consist of two major parts, a header section and a data section. Each item in the header section is three lines long; each item in the data section is two lines long. This is the garbage you saw when you loaded the file into the AppleWorks word processor as an ASCII text file.*

The header section of a DIF file looks something like this:

```
TABLE
0,1
""
VECTORS
0,2
""
TUPLES
0,26
""
DATA
0,0
""
```

This is the minimum amount of information you'll find in a DIF file header. The protocol allows programs that create DIF files, such as ***VisiFile,*** *to insert additional header items between the 3-line TABLE item and the 3-line DATA item. The protocol says that programs reading DIF files should ignore these extra pieces of information if they don't recognize them.*

AppleWorks, however, refuses to read DIF files that have anything in the header other than the minimum amount of information. This is a bug that should be fixed.

In the meantime, the solution is to load the DIF file into the word processor (tell the word processor it's a standard ASCII text file, not a DIF file, just as you did before) and delete all the 3-line items in the header except the four shown above. Then print the file back into an ASCII text file. This new text file really has the internal format of a DIF file, of course, and will load directly into the spreadsheet or data base by way of the "make a new file from a DIF file" route.

I wrote up a complete description of the internal format of DIF files, as well as Applesoft subroutines for reading and writing them, in the February 1984 ***Softalk****, page 65, if you'd like to find out what all that garbage was.*

Open-Apple

is written, edited, published, and

Business Consultant Richard Barger
Technical Consultant Dennis Doms
Circulation Manager Sally Tally

Open-Apple
P.O. Box 7651
Overland Park, Kansas 66207 U.S.A.

ISSN 0885-4017. Published monthly since January 1985. World-wide prices (in U.S. dollars; airmail delivery included at no additional charge): $24 for 1 year; $44 for 2 years; $60 for 3 years. All back issues are currently available for $2 each; a bound, indexed edition of Volume 1 is $14.95. Index mailed with the February issue. Please send all subscription-related correspondence to:

Open-Apple
P.O. Box 6331
Syracuse, N.Y. 13217 U.S.A.

Subscribers in Australia and New Zealand should send subscription correspondence to ***Open-Apple***, c/o Cybernetic Research Ltd, 576 Malvern Road, Prahran, Vic. 3181, AUSTRALIA.

Open-Apple is available on disk for speech synthesizer users from Speech Enterprises, P.O. Box 7986, Houston, Texas 77270 (713-461-1666).

Source Mail: TCF238 **CompuServe: 70120,202**